SCHEDULING PROBLEMS

by
Aamir Kudai

A thesis presented in partial fulfillment of requirements
of the the Sally McDonnell Barksdale Honors College
at the The University of Mississippi

Oxford
May 2018

Approved by

Advisor: Dr. Dawn Wilkins

Reader: Dr. Conrad Cunningham

Reader: Dr. Yixin Chen

ABSTRACT

Manufacturing industry is growing exponentially. The need of using algorithms and computational techniques to enhance processes is increasing every day. Algorithms help us solve almost all kind of computational problems. Not only choosing the right algorithm for a problem is important but also optimizing its time and space efficiency is crucial. BorgWarner Transmission Systems located in Water Valley, Mississippi is one among the leading manufacturing companies. This paper will demonstrate a real-world audit scheduling problem happened at BorgWarner and the techniques used to solve it. A gentle introduction to some of the heuristic algorithms such as Genetic algorithm, Randomized algorithm, and Greedy algorithm is also discussed in this paper. A description of how the problem is modelled to fit through these approaches and how good they work in each approach, as well as their time and space consumption is been analyzed in this paper.

DEDICATION

For Razi Kudai, Noori Kudai and Anas Kudai

ACKNOWLEDGEMENTS

The completion of this study could not have been possible without the expertise of my thesis advisor, Dr. Dawn Wilkins. I would also like to thank Mr. Anthony Weaver for assigning me such a challenging project and assisting me throughout the process of problem solving.

A debt of gratitude is owed to Ms. Amy Chrestman for giving me the opportunity to work with BorgWarner and to believe in me, when I did not believe in myself. A big thanks to the entire BorgWarner team for giving me a chance to gain knowledge, professionalism and experience.

Finally, I would like to acknowledge with gratitude, the support and love of my family - my parents and my brother.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BW BorgWarner

L1 Level 1 Employees

L2 Level 2 Employees

L3 Level 3 Employees

GA Genetic Algorithm

CHAPTER 1

INTRODUCTION

When we observe around us, we find a continuous series of activities happening related to a variety of things, some that affect us some that do not. But they definitely are meant for a purpose to achieve something, and that why people spend their resources in doing them. If we pick any one among these activities, its certain that we think about an idea that it could be done better, if some component, method or way of this activity is changed or altered. This is how I came across one of the several activities going around a plant in a company named BorgWarner located in Water Valley, Mississippi.

BorgWarner Inc. is an American worldwide automotive industry components and parts supplier. It is primarily known for its powertrain products, which include manual and automatic transmissions [BorgWarner Inc. (2013)]. Maintaining quality is most important for BorgWarner and they do the most they can to maintain micron level accuracy in their production to maintain high quality. To maintain quality, each process was to be examined and periodically monitored. Audits used to be a powerful tool for BorgWarner's quality control department. Going into little details of the audit process, the auditing process was laid throughout the plant, covering various layered/nested areas of the plant. Some audits covered a smaller area whereas some covered a relatively larger area. These audits were performed by different types of employees. The problem was the audits were not evenly spread distributed the employees through out the plant. Some employees complained about doing too many audits, and some were never picked to do one. Mr. Anthony Weaver, Quality Manager

of BorgWarner, used to spend hours at the end of every month to make the schedule as appropriate as he can. However, he used to almost use the same schedule as the previous month and used to tweak it a little. This also wasn't efficient, firstly because it wasn't 100% perfect schedule and secondly because most of the employees were assigned the same audits every month. I came across Anthony in the middle of my internship and that's how we began to work towards resolving this issue.

Algorithms are useful tools that you can use to solve a variety of problems. Especially if the problem has to deal with some sort of data available on a computer. Formally, an algorithm is any well-defined procedure for solving a given class of problems. Ideally, when applied to a particular problem in that class, the algorithm would yield a full solution [Van Huyssteen and Library (2003)]. The key to solve this problem was to find the right problem. I did not have enough knowledge about which one to choose, I needed assistance and Dr. Dawn Wilkins, Department chair of computer and information science, The University of Mississippi, was my helping hand. I did figure out that the problem was a scheduling problem but wasn't sure if it was a decision problem or optimization problem. Optimization can be defined as finding solution of a problem where it is necessary to maximize or minimize a single or set of objective functions within a domain which contains the acceptable values of variables while some restrictions are to be satisfied Rao et al. (2016). Whereas a decision procedure is an algorithm that, given a decision problem, terminates with a correct yes/no answer. More details about the description of the problem and how I solved it will be discussed later in this paper [Kroening and Strichman (2016)]

CHAPTER 2

AUDIT SCHEDULING PROBLEM

Before moving on to discuss what the audit scheduling problem is, It is important to know what constraint satisfaction problem is. The Audit scheduling problem closely resembles to a constraint satisfaction problem.

2.1 Constraint Satisfaction Problem

A constraint satisfaction problem is a problem in which a set of values needs to be assigned to a set of variables obeying all constraints specified in the problem. In a constraint satisfaction problem (CSP), the aim is to find an assignment of values to a given set of variables, subject to specified constraints [Bulatov (2011)]. Formally, a constraint satisfaction problem is defined as a triple $\langle X, D, C\rangle$ where,

$X = \{X_1, X_2, \ldots, X_n\}$ is a set of variables

$D = \{D_1, D_2, \ldots, D_n\}$ is a set of the respective domains of values, and

$X = \{X_1, X_2, \ldots, X_n\}$ is a set of constraints.

Each variable X_i can take on the values in the nonempty domain D_i. Every constraint $C_i \in C$ is in turn a pair $\langle t_i, R_i\rangle$, where $t_i \subset X$ is a subset of k variables and R_i is a k-ary relation on the corresponding subset of domains D_j. An evaluation of the variables is a function from a subset of variables to a particular set of values in the corresponding subset of domains. An evaluation v satisfies a constraint $\langle t_i, R_i\rangle$ if the values assigned to the variable t_i satisfies the relation R_j.

An evaluation is consistent if it does not violate any of the constraints. An evaluation is complete if it includes all variables. An evaluation is a solution if it is consistent

and complete; such an evaluation is said to solve the constraint satisfaction problem [Russell and Norvig (2016)] .

2.2 The Scheduling Problem

The Audit scheduling problem at BorgWarner was a problem similar to a Constraint satisfaction problem where a set of auditors (values) are assigned to a set of audits (variables) satisfying the constraints.

2.2.1 Variables

In this problem the set of variables are audits. Audits are of three types: - Daily audit, Weekly audit and Monthly audit.

1. **Daily Audits** are very basic level audits which need to be done daily within a small area.

2. **Weekly audits** are audits that need to be done every week across a larger area compared to Daily audits.

3. **Monthly audits** need to be done every month and they cover the largest area of the plant.

This way the area covered in the audits are nested in a Monthly $\rightarrow$ Weekly $\rightarrow$ Daily fashion.

2.2.2 Values

The set of values that should be assigned to variables are Employees. The employees or auditors also are categorized into three types named as: - Level 1, Level 2 and Level 3.

1. Level 1 employees are **Team leaders** who audit their own stations and nearby areas. **(Daily Audit)**

2. Level 2 employees are **Engineers** who audit the entire plant line comprising of stations. **(Weekly Audit)**

3. Level 3 employees are **Quality Managers** and Supervisors who audit multiple plant lines. **(Monthly Audit)**

2.2.3 Constraints

The Constraints that need to be satisfied while assigning the auditors to the audits will be explained in this section. We are going to solve the problem of Daily audit assignment in this paper. The rest of the Audits (Weekly and Monthly) are illustrated to demonstrate the entirety of the auditing process.

2.2.3.1 Eligibility

This subsection explains which auditor is eligible to perform which type of audit.

1. A Level 1 Employee can only perform only **Daily Audit**

2. A Level 2 Employee can perform **Weekly** and **Daily Audits**

3. A Level 3 Employee can perform **Monthly, Weekly** and **Daily Audits**

AUDITORS ↓	**Daily Audit**	Weekly Audit	Monthly Audit
Level 1	**YES**	NO	NO
Level 2	**YES**	YES	NO
Level 3	**YES**	YES	YES

Table 2.1. Performable Audits table.

Weekly audits and Monthly audits have been included in the eligibility section to have a picture of the auditing process. The focus of this explanation is to show that Daily audits can be performed by all types of employees but should be limited by the frequency of allotment per employee category which will be shown in the next subsection.

2.2.3.2 Frequency of Employee Assignment

The List below demonstrates the frequency of employee participation in Daily Audits per category of employee throughout the schedule. The schedule is usually generated for a month at a time.

AUDITORS	Frequency of Participation in each daily audit
Level 3	Once per month
Level 2	Once per week
Level 1	Once per Day (If not occupied)

Table 2.2. Frequency of Participation.

2.2.3.3 Load per Employee

A single Employee (Auditor) should not perform more than 2 audits of any type per day.

2.2.3.4 Solution

A solution in which above the values (Employees) are assigned to variables (Auditors) satisfying all of the constraints listed above. The desired output would be in a list format with column headers such as auditor, audit, date.

AUDIT	AUDITOR	DATE
Ford AODE - 171 AODE Machining	Goad, Travis	01/01/2018
Chrysler - 282 41TE-NVH Assembly	Minor, Bernard	01/01/2018
Molding - 301 Ransohoff Washer	Taylor, Edmond	01/02/2018
Receiving/Shipping Department	Boyd, Barry	01/02/2018
Ford 6F Mid - 230 6F Winder assembly	Johnson, Shirley	01/02/2018
GM F4 - 279 (2) F4 Assembly	Lindley, Tristen	01/03/2018
GM MDA GF9 - MDA GF9 Assembly	Richardson, Walter	01/04/2018
IMV - IMV - Layered	Taylor, Edmond	01/05/2018
Molding - 301 Ransohoff Washer	Goad, Travis	01/05/2018

Table 2.3. Structure of Sample output.

The above names used are dummy names for auditors and audits. The output is not an actual solution to the scheduling problem which means that it does not represent the satisfiability of constraints but serves to show the structure of output.

2.3 Formal Description

Let $T = \{T_1, T_2, T_3, \dots T_n\}$ be a finite set of Audits

Let $L_1 = \{A_1, A_2, \dots A_p\}$, $L_2 = \{B_1, B_2, \dots B_q\}$ and $L_3 = \{C_1, C_2, \dots C_r\}$ be finite sets of Employees such that $L_1 \cap L_2 = \emptyset$, $L_2 \cap L_3 = \emptyset$, $L_1 \cap L_3 = \emptyset$

Let X denote the set of all assignment of tasks to employees

Elements $x \in X$ may be written as $(i \times j)$ matrices, in which column i represents days, row j represents Audits, and every (i, j) assignment of Employee L represents that audit i is assigned to him on day j:

$$\mathbf{X}_{i,j} = \begin{pmatrix} A_8 & B_4 & A_4 & A_1 & C_2 & A_5 & B_5 & A_4 & A_2 & A_2 \\ A_2 & A_5 & A_1 & B_3 & A_4 & A_3 & C_4 & A_4 & B_3 & A_3 \\ A_2 & B_1 & C_4 & A_6 & B_7 & A_8 & B_2 & A_4 & A_7 & B_5 \\ A_4 & C_1 & A_2 & B_4 & A_1 & A_3 & B_1 & A_4 & A_6 & C_7 \end{pmatrix}$$

such that :

1. $\forall i$ (Row), there $\exists!$ element from set L_3 for every 20^{th} interval of each Column.

2. $\forall i$ (Row), there $\exists!$ element from set L_2 for every 5^{th} interval of each Column.

3. $\forall \mathbf{X}_{i,j}$, that is empty, an element from set L_1 will be assigned to $\mathbf{X}_{i,j}$.

4. $\forall j$ (Column), there are atmost 2 occurences of any $l \in (L_1 \vee L_2 \vee L_3)$.

2.4 Importance of Audit Scheduling at Borgwarner

The Quality System of any plant focuses on how well and how efficient the plants end products are shipped. On each operation throughout the plant, whether it is machining of parts to assembly, there is a test stand which tests if the operations performed on the stand maintained the standard of the part or have made scrap. Even though the Test stand has the responsibility to check whether the part produced was good or bad the auditors take a bigger responsibility of auditing the test stand plus other manufacturing areas that are important throughout the plant.

2.5 The need for Scheduling

The Continuous quality improvement (CQI-8) is a manual that contains guidelines for quality control in the automotive manufacturing industry [CQI2013automotive]. CQI-8 explains guidelines instructed by American Industry Action Group concerning the execution, organization and implementation of audits. The Automotive Industry Action Group (AIAG) is a non-profit trade association that develops solutions for

the automotive industry [AIAG 2018 industry]. Scheduling audits appropriately is one of the guidelines stated in the CQI-8. Appropriately following CQI-8 guidelines not only helps automotive companies enhance quality but also helps them maintain industrial standards.

Moreover, customer specific requirements also tend to force having a consistently varying schedule that maintains employees to audit each component of the plant in order to produce exceptionally accurate parts, and parts that fall within standards. "We need to do the LPA process correctly to fulfill the customer requirements, The customers are the reason we stay in business", says Anthony Weaver, the quality manager of BorgWarner. Auditing guarantees the process does the right things to provide quality parts to the customers. Because observing, supervising and correcting something wrong is the most efficient way to fulfill standards. The bigger impact is that this is the leading indicator, it highlights where is the risk of sending a defective part to a customer. What happens if we don't meet the constraints? We'll be spending resources on no-value added audits. A customer could give us a corrective actions and we would not be in business with the customer. We have a compliance side and a value added side to do the audits and scheduling brings them both together.

2.6 What was wrong in the way it was originally scheduled?

The audits were scheduled by the Quality Engineers manually, hand picking individuals and assigning them audits on particular days. They used to spend hours on scheduling but were yet not able to fulfill all constraints. What ended up was a standard monthly schedule which could fulfill the most constraints but could still not have 100% of constraints satisfied plus the rollover schedule had the same people doing the same audits every month. This leads to two problems, first was the employees being upset about doing the same audit over and over and second was the constraints of schedule not being satisfied. Mr. Anthony Weaver stated that he spends more

than 3 hours to schedule an incorrect schedule.

CHAPTER 3

INITIAL APPROACH

3.1 Initial Idea

The purpose of the initial approach was to generate a schedule that satisfies all constraints. The idea began with creating a 2 Dimensional array such that rows of the array represent audits and columns of the array represent days. This way assigning an individual to a cell will represent the specific audit as well as on what day he or she is going to perform that audit.

The representation into a 2D array (also called matrix) not only helps easily populating the auditors to audits, but also helps in checking constraints. Figure 3.1. shows the representation of the 2 Dimensional matrix.

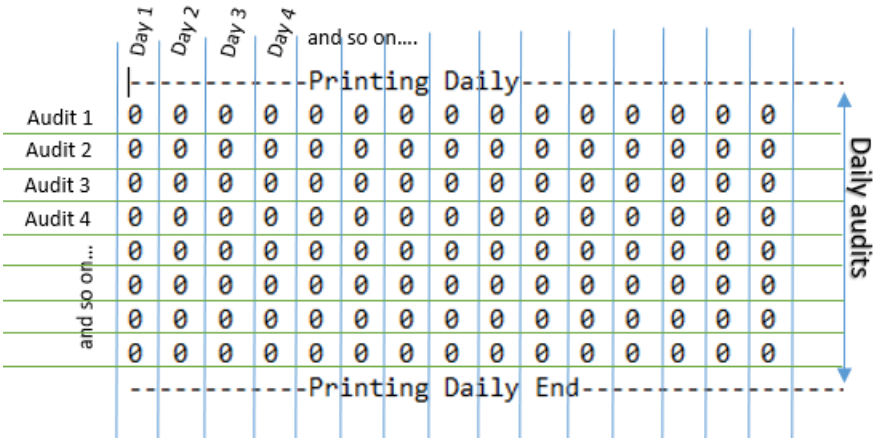

Figure 3.1. The Initialization of 2 Dimensional array as matrix

3.2 Populating Auditors into the array

Once the structured space is created, the auditors need to be assigned to the audits

3.2.1 The Accommodation of Level 1 Employees

To have a better visualization, the assignment of auditors and to also keep the solution compact, let Level 1 auditors be A1, A2, A3 ... An, where n is the count of level 1 auditors. This makes the solution appear tightly packed and symmetrical. The idea of populating auditors was to iterate through each cell of the array and pick a random auditor from A1 to An on each cell. Figure 3.2. below shows the 2 dimensional array after populating with random level 1 employees.

```
A3 A1 A4 A2 A2 A3 A2 A4 A4 A2 A0 A2 A2 A2 A3 A0 A1 A2 A3 A1
A0 A4 A3 A3 A3 A4 A2 A2 A2 A4 A1 A4 A3 A2 A3 A1 A1 A3 A2 A3
A2 A1 A2 A2 A1 A4 A3 A2 A3 A4 A0 A0 A4 A2 A4 A2 A3 A4 A2 A3
A4 A0 A3 A3 A4 A3 A0 A4 A4 A1 A1 A1 A4 A2 A2 A3 A2 A0 A4 A2
A3 A2 A0 A4 A3 A1 A1 A3 A2 A1 A3 A2 A4 A1 A4 A0 A2 A1 A0 A1
A0 A4 A0 A1 A4 A3 A0 A0 A0 A1 A4 A0 A3 A4 A0 A4 A4 A4 A1 A4
A4 A0 A0 A4 A0 A2 A1 A2 A0 A3 A1 A3 A3 A1 A2 A2 A4 A0 A3 A2
A1 A3 A0 A3 A2 A1 A2 A0 A0 A3 A1 A3 A2 A2 A0 A2 A1 A2 A1 A3

-------------Below shows days with conflicts---------------
0  0  1  1  0  1  1  1  1  1  1  0  1  1  0  1  1  0  0  1
```

Figure 3.2. Populating L1's and computing conflicts on each column

To determine the constraint which calculates the load on auditor per day, there is a fault checker which checks how many occurrences of each employee happens on a particular day (Column). After Scanning through the solution, the fault checker returns an array which shows which day has violated the constraint of load per employee (no more than 2 audits per day). A 0 represents there was no fault and a 1 represent there was one or more faults on that column (DAY). After the random assignment and fault checking is done. The program goes through the Matrix and regenerates faulty columns randomly the same way as it did originally. The difference is that now

it only regenerates the faulty columns. The faults are reduced or sometimes remain the same. The loop keeps going on and on until it finds a solution with all columns having 0 faults. The solution keeps randomly regenerating columns that are faulty until the stopping condition is reached.

3.3 The Accommodation of Level 2 Employees

Since level 2 auditors need to share the responsibility of doing each Audit once per week. The initial Idea was to mark one day of the week for Level 2 to perform the audit. The chosen day was Friday or the end of the week. Audits are needed to be completed by the assigned date, which means that the day that is assigned to them by the system is a due date. Let Level 2 auditors be B1, B2, B3 ... Bp, where p is the count of level 2 auditors. Having the due date of Friday gives enough time for level 2 employees to perform the audit. The already assigned level 1 auditors are overwritten by level 2 auditors. This reduces the load on level 1 auditors. Figure 3.3. shows the matrix after assignment of level 2 auditors.

A1	A3	A0	A4	B0	A0	A3	A4	A4	B2	A4	A0	A2	A4	B4	A3	A4	A4	A1	B4
A0	A0	A1	A1	B2	A0	A4	A4	A4	B6	A1	A1	A1	A2	B2	A4	A3	A0	A0	B4
A1	A0	A2	A2	B0	A2	A3	A3	A2	B5	A1	A3	A4	A3	B6	A4	A2	A1	A0	B1
A3	A1	A0	A4	B2	A1	A0	A3	A0	B0	A4	A4	A1	A4	B4	A3	A1	A3	A2	B3
A2	A4	A2	A2	B5	A3	A0	A0	A3	B0	A2	A3	A2	A1	B5	A2	A1	A0	A4	B2
A4	A2	A4	A0	B6	A4	A4	A0	A2	B6	A0	A1	A4	A0	B0	A2	A4	A3	A2	B2
A4	A2	A4	A1	B6	A4	A2	A1	A1	B2	A3	A0	A0	A1	B3	A0	A2	A2	A4	B3
A2	A1	A3	A0	B3	A1	A1	A2	A1	B3	A2	A4	A0	A2	B6	A1	A0	A2	A3	B5

Figure 3.3. L2's populated on every end of working week

We can observe that at every interval of 5, we see a Level 2 person performing the audit. The fault checker treats the newly assigned level 2 columns the same way as it did for level 1. It goes through all the columns and marks whether any employee has more than 2 occurrences. If then regenerates the column, including the level 2 assigned column if the load on any employee is more than 2.

3.4 The initialization of Level 3

The same way as level 2 auditors had to perform daily audit once a week, level 3 auditors need to perform it once per month. To accommodate the level 3's without changing the logic, the last day of the schedule was chosen to be the day when Level 3 auditors will share the responsibility. As it would be the end of the month the level 3's will get sufficient time to perform the audit.

After a few running and testing the program a few times, I observed that for days when the total number of days was even, let's say for instance if it was 20. The last days that would have level 2's assigned already, would be wiped off by Level 3's.

To solve this problem, the day for level 3 auditors needs to be moved accordingly to avoid overwriting the level 2 auditors. To avoid this, for every schedule that has an even number of total days, the schedule will allocate the second last day instead of last day of the schedule. This will make sure the level 2 assignment is not overwritten. For schedules with total number of days odd will the allocation will remain to be the last day of the month.

Algorithm 1 Pseudocode

```
1: if TotalDays%2 == 0 then
2:     IndexForMonthly = (TotalDays − 2)
3: else
4:     IndexForMonthly = (TotalDays − 1)
```

This makes the schedule adjust accordingly as per the selection of days.
For instance, if the total number of days was 20 (even) then the schedule will look like the Figure 3.4.

```
A1 A1 A0 A2 B2 A1 A0 A0 A1 B4 A4 A2 A0 A2 B1 A4 A0 A2 C2 B5
A1 A3 A3 A0 B4 A3 A2 A4 A4 B0 A1 A0 A4 A0 B2 A1 A0 A2 C3 B6
A4 A3 A3 A3 B5 A4 A1 A3 A3 B0 A0 A2 A1 A3 B0 A2 A1 A4 C1 B0
A3 A2 A4 A2 B3 A3 A0 A2 A3 B5 A2 A3 A2 A4 B2 A3 A3 A4 C4 B3
A4 A2 A2 A0 B6 A2 A4 A2 A4 B5 A1 A4 A1 A3 B0 A1 A4 A0 C0 B4
A0 A4 A1 A3 B3 A0 A3 A1 A1 B6 A4 A1 A3 A2 B3 A2 A3 A3 C4 B1
A3 A0 A1 A4 B6 A1 A4 A0 A2 B3 A0 A1 A4 A1 B4 A0 A4 A3 C2 B3
A0 A1 A4 A4 B1 A0 A3 A3 A2 B6 A2 A4 A2 A0 B6 A3 A2 A0 C1 B2
```

Figure 3.4. Schedule for 20 days

And similarly when total number of days is 21, then the schedule looks like Figure 3.5.

```
A3 A2 A4 A4 B6 A0 A1 A2 A4 B0 A3 A1 A4 A4 B5 A1 A2 A4 A0 B1 C3
A1 A4 A4 A0 B6 A3 A0 A0 A1 B0 A2 A0 A0 A3 B0 A0 A3 A1 A2 B3 C0
A3 A0 A0 A4 B5 A2 A4 A0 A0 B5 A1 A2 A2 A2 B5 A0 A1 A0 A0 B1 C3
A2 A3 A2 A0 B2 A4 A3 A1 A2 B2 A4 A1 A3 A1 B3 A3 A3 A4 A4 B6 C2
A0 A4 A0 A3 B3 A0 A4 A2 A1 B4 A3 A3 A1 A4 B4 A3 A4 A2 A4 B4 C2
A1 A3 A3 A2 B5 A3 A1 A1 A2 B6 A2 A4 A0 A0 B4 A4 A1 A3 A1 B5 C1
A2 A1 A1 A2 B2 A4 A3 A4 A3 B2 A0 A4 A4 A1 B1 A2 A4 A2 A1 B5 C4
A0 A2 A3 A3 B4 A1 A0 A4 A3 B3 A1 A0 A1 A0 B2 A2 A2 A3 A3 B4 C4
```

Figure 3.5. Schedule for 21 days

3.5 The issues with this methodology

1. The biggest drawback of this solution is the probability of randomly getting a column and having it randomly get a column that has no faults. While testing the solution for a small number of audits, the program would reach a solution in a reasonable amount of time. But after substituting the actual number of audits (e.g. 44 Audits), the probability of getting a conflict free day reduced considerably. The solution would run for several minutes and sometimes keep looping to find a solution randomly but could not find one in less than 30 minutes.

2. The second drawback was the allotment of permanent days for the level 2's and

level 3's throughout the schedule. It not only makes the schedule less evenly distributed and fair, but also put the constraint of having at least more the half the total number of level 2's and level 3 employees than the total number of audits. As to accommodate a column (Day) with let's say 4 audits; we at least need 2 or more employees to avoid conflicts.

3. Another drawback is that the solution is considered to be a schedule for a month. The approach needs modifications and manipulations to be able to generate a schedule for more than a month.

CHAPTER 4

FINAL APPROACH

4.1 Overview

The initial idea of randomization discussed in the previous chapter could satisfy constraints and could also yield a solution. However, it lacked many parameters such as sparsity and evenness. The prime reason for this was the concentration of higher level employees on one pre-determined day. The key to solve this was to use different days during a month instead of the same day, plus maintain the constraint about no employees doing more than 2 audits per day. To get rid of the fault column repair rule and to put forward a method that could get a feasible solution in less time was needed, a greedy approach was the answer to this.

4.2 Greedy Approach

A greedy approach as the name suggests, is an approach that strives towards reaching a solution in a greedy way. The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached [Levitin (2012)]. In this algorithm for each decision point, the choice that seems the best is chosen. The Greedy approach makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution [Cormen et al. (2009)]. This heuristic approach does not always produce an optimal solution but in some decision problems like the problem discussed in this paper, they work perfectly fine. In decision problems there is a yes or no answer to the solution. One solution in our problem differs from another solution but does not give an optimality measure based on which both can be differentiated.

4.3 The Algorithm

We know that the Level 3 auditors share the load of daily employees once per month and Level 2 auditors share the load once per week; setting them first made more sense as once they are being assigned audits on certain days, the load on those days for level 1 auditors is being eliminated as the level 1 auditors do not have to perform audits on those days.

The assignment of each category of auditors one after the other was carried out by finding a random day within days and finding the right auditor for that particular day. Doing this until it finds the right fit satisfying constraints.

The basic Idea (Rule of thumb) used for this was:

1. Find a random spot (within the permissible range)
2. Pick a random person (from the list of auditors)
3. See if the auditor can accommodate an audit (Constraint 5)
4. If he or she can accommodate an audit $\rightarrow$ make the assignment, If not go back to step 2
5. Try it n times (where n is the total number of employees of that category)
6. If not found, go back to the step 1

The technique can be demonstrated clearly with the Pseudocode, as given in Algorithm 2.

4.4 Implementation

Similar to the initialization of 2 D array as was done in the previous approach, we initialize a matrix of having columns that represents days and rows that represent audits. As shown in the Figure 4.1.

Algorithm 2 Pseudocode

```
1: for all audits do
2:     do                                                    ▷ For all Intervals
3:         StoppingCondition = 0
4:         exitDo = 0
5:         do
6:             spot = Random.IntervalRange
7:             if day[spot] = null then
8:                 person = Random.EmployeeList
9:                 if person.AuditsPerDay < 2 then
10:                    day[spot] ← person
11:                    exitDo = 1
12:        while exitDo ≠ 1 OR StoppingCondition > 10000
13:    while End of Intervals
```

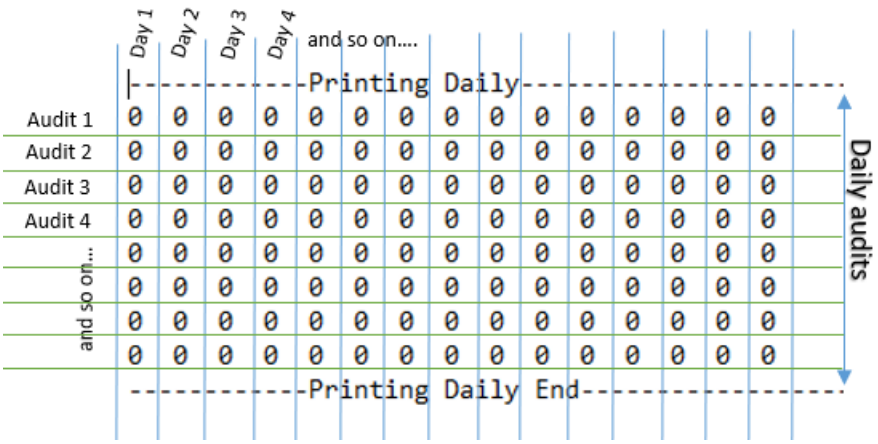

Figure 4.1. The initialization of 2 Dimensional array as matrix

4.4.1 Level 3 assignment

Since L3's are supposed to do Daily audits once a month. We make sure that for every interval of column (days) there exists exactly one level 3 auditor, for all daily audits. The interval of 20 should extend iteratively increment by 20, until the end of the schedule is reached. Figure 4.2. shows assignment of level 3 auditors.

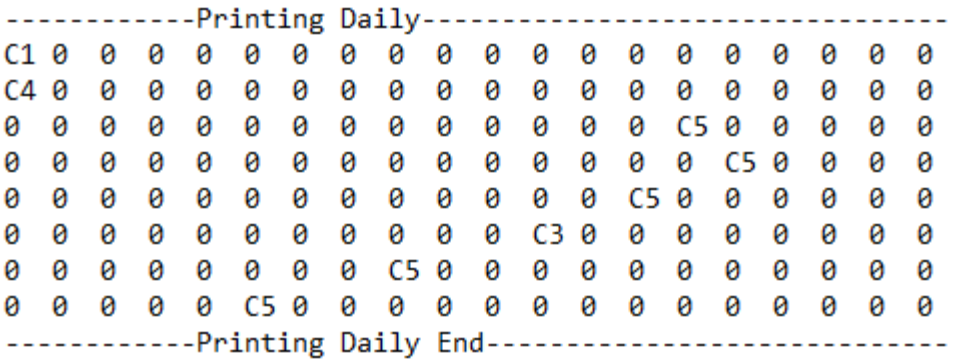

Figure 4.2. Assignment of Level 3

To do this, we generate a random number from 0 to 20 for the first interval. If the number of days has 20 more days after the first interval we consider it as a second interval from 20 to 40. The application tries to evenly divide the days into intervals of 20 and truncates the remaining days. Going back to the generation of a random number from 0 to 20. Once that is done, a random level 3 auditor is picked by generating a random number that represents the index of the auditor. After we have found the spot (cell of the 2d array) and the auditor to be placed, we check whether the employee has exceeded the workload of 2 audits for that day or the employee is eligible to accommodate an audit. This is done by vertically going through the columns (as columns represent days) and counting the number of occurrences of that employee on that day. If the auditor has more than the maximum audits (2 audits/ day) an auditor can do per day, the employee is not eligible to be assigned the audit on that day. We randomly pick another employee and perform the check. This is done for the total auditors present in the list. If still an auditor could not be assigned on that spot, We go back to the step of picking a spot and start over. This is also done until the stopping condition, which is the interval length since the interval length is all possible spots that are available. If an auditor can accommodate an audit on the randomly assigned day, he is given the audit and the assignment is finalized. Please see the Algorithm 2 for clearer view.

4.4.2 Level 2 Assignment

```
------------------Printing Daily-----------------------------
C1 0  0  0  B1 0  0  0  0  B7 0  0  0  0  B2 0  0  0  B3 0
C4 0  0  0  B7 0  0  B4 0  0  0  0  0  0  B7 0  0  B5 0  0
0  0  0  0  B2 0  0  0  0  B1 0  0  B7 0  C5 0  B4 0  0  0
0  0  0  B4 0  0  0  0  0  B4 B3 0  0  0  0  C5 0  0  B6 0
B4 0  0  0  0  B5 0  0  0  0  0  0  B1 C5 0  0  B6 0  0  0
B3 0  0  0  0  0  0  0  0  B7 B7 C3 0  0  0  0  0  0  B7 0
0  0  0  B7 0  0  0  0  C5 B2 0  0  0  0  B4 0  0  0  0  B6
0  B1 0  0  0  C5 0  B3 0  0  B3 0  0  0  0  0  0  B2 0  0
------------------Printing Daily End-----------------------
```

Figure 4.3. Assignment of Level 2

L2 auditors need to take responsibility of doing daily audits once per week. Figure 4.3. shows the assignment of Level 2 auditors. We use the exact same method as we did to assign L3's every month but in this case the interval is 5 instead of 20. Because a working week comprises of 5 days and our list of days does not encounter holidays, having a L2 on every interval of 5 i.e. 0 to 4, 5 to 9, 10 to 14 and so on, works great and satisfies the constraint. We use the same strategy of finding a random spot within interval (0 to 4) and finding a random employee, and assigning it the particular spot, if he can accommodate an audit or starting over if he cannot accommodate.

4.4.3 Level 1 Assignment

```
-----------------Printing Daily-----------------------------
C1 A4 A2 A5 B1 A3 A1 A1 A3 B7 A1 A3 A1 A1 B2 A3 A2 A1 B3 A2
C4 A2 A1 A1 B7 A1 A3 B4 A3 A2 A3 A5 A3 A3 B7 A3 A2 B5 A1 A4
A4 A4 A4 A5 B2 A5 A1 A4 A1 B1 A1 A2 B7 A3 C5 A4 B4 A5 A4 A2
A3 A1 A3 B4 A4 A1 A3 A2 A4 B4 B3 A3 A2 A5 A1 C5 A5 A1 B6 A4
B4 A1 A2 A1 A4 B5 A5 A4 A2 A4 A3 A1 B1 C5 A2 A4 B6 A3 A3 A3
B3 A5 A3 A3 A3 A3 A2 A5 A2 B7 B7 C3 A2 A4 A5 A2 A5 A4 B7 A1
A2 A5 A4 B7 A5 A5 A4 A5 C5 B2 A5 A2 A4 A1 B4 A2 A1 A2 A2 B6
A4 B1 A1 A3 A1 C5 A5 B3 A4 A3 B3 A4 A3 A5 A4 A5 A1 B2 A4 A1
-----------------Printing Daily End-------------------------
```

Figure 4.4. Assignment of Level 1

After L3's and L2's are assigned Daily audits, the remaining spots are filled with daily auditors by iterating thorough the 2d array. Each spot is filled by a random L1 auditor and it is made sure that the already assigned higher level auditors are not overridden. It is also checked whether level 1 auditor can accommodate an auditor before assigning him the audit on a particular day.

4.4.4 The sequence of levels

The occurrence of a level 3 employee for every audit (Row) is going to be once in every 20 day. Which means that in every 4 weeks i.e. 20 working days a level 3 is responsible to perform the daily audit. Similar to that the occurrence of a level 2 happens every week i.e. every 5 days (working days).After level 2's and level 3's take responsiblity of doing some daily audits, the rest will be taken care by the level 1 employees. In this way, the load for level 1 employees is been reduced. Because of this nature of our problem, the sequence of assigning level 3's first, followed by level 2's followed by level 1's made more sense as in this way the spots where the L3's and L2's are occupied, L1's will no longer be responsible to do that audit on that particular day.

4.4.5 Satisfying Constraints

With the intention of finding the right day and the right auditor, the approach satisfies all constraints. while it searching for the right spot (cell), it performs the search within the range interval. Example - while searching for L3 to assign once a month, it looks for a random spot within intervals if 20. Similarly for L2's it looks for a spot within intervals of 5. The rest of the audits are done by L1s. This makes sure there is atleast someone doing the audit satisfying the following constraints.

1. Each Daily Audit should be happen Daily by eligible employees.

2. Each Daily Audit should be done by a L3 once per month.

3. Each Daily Audit should be done by a L2 once per week.

Checking the number of occurrences per day which is performed before assigning an employee to an audit makes sure that no employee is doing more than a certain number of audits allowed per day. This satisfies the following constraint.

1. No Employee should do more than a certain number of audits per day.

CHAPTER 5

GENETIC ALGORITHM

5.1 What is Genetic Algorithm

A genetic algorithm is a search heuristic that works on the principle of "survival of the fittest". The genetic algorithm attempts to find a good (or best) solution to the problem by genetically breeding a population of individuals over a series of generations. In the genetic algorithm, each individual in the population represents a candidate solution [Koza (1992)]

This algorithm is useful when the search space is very large, or the problem is too complex in other words the problem is NP hard. The algorithm basically contains five phases such as: Initial population, Fitness function, Selection, Crossover and Mutation. Please note that the selection phase can be eliminated to make the generation diverse. Also, a termination point is essential for genetic algorithm, in case it can't find the solution in suitable number of iterations. The process starts with randomly populating chromosomes or individuals with genes. A set of all randomly populated individuals is called a population. Each individual or chromosome is a solution to the problem we want to solve. Each individual contains a set of variables/genes/features. After a population is formed, a fitness function computes the fittest individual among the population.

5.1.1 Fitness of Solution

A fitness function is kind of like an evaluator that examines every individual based on a criteria which is problem specific and gives the individual a fitness score. Every

individual in the population is assigned, by means of a fitness function, a measure of its goodness with respect to the problem under consideration. This value is the quantitative information the algorithm uses to guide the search [Sivanandam and Deepa (2008)]. Let us take an example of a problem in which the goal is to achieve all one's in an array of size 10, where the possible entries to the array can be a 0 or a 1. In this case, criteria for the fittest is to have the maximum number of 1's. Notice the fittest individual here is the second Chromosome in the table having a fitness of 10. The table shows an individuals being randomly initialized with 0's and 1's and a column which shows the fitness function for each chromosome.

Individual	Fitness
1101101001	6
1111111111	10
0000100100	2

Table 5.1. fitness table.

5.1.2 Reproduction (Crossover)

The reproduction or creation of new individual is done by doing crossover. In crossover, a crossover point (index of the array) is randomly chosen between two individuals (in our case between the 1st fittest and 2nd fittest) exchanges the subsequences before and after that point between two chromosomes to create two offspring [Mitchell (1998)]. All Variables or genes before the crossover points are exchanged by two individuals and new offspring are formed.

5.1.3 Mutation

After offspring are formed, the next phase is mutation. To maintain diversity and to avoid convergence, some variables or genes are flipped in the offspring [Mitchell

(1998)]. Once we have another set of individuals, which a mixture of old and some newly created offspring, the fitness function again evaluates the fittest individuals and the process goes on until we find the most fit individual or the individual with the highest fitness score (solution to the problem).

5.1.4 Stopping Condition

Usually a predetermined number of generations is specified which is used to cut of the search in case if the solution is not found within that range.

5.2 Implementation

This section describes about how the problem was solved using Genetic Algorithm. I have used Genetic Algorithm from the Genetic Algorithm for Java Basics [Jacobson and Kanber (2015)] and modified the essential to parts of the GA in order for it to adapt to the problem discussed in this thesis.

5.2.0.1 Gene Representation

Due to the way Genetic algorithm is structured in the book, A gene is represented as a 2 Dimensional array. In order to make the problem adapt to the system, a Gene or a solution is represented in terms a 2D array instead of matrix by breaking the rows of the matrix and appending each row to another. This makes the matrix a long array. This representation not only makes it easier to blend the audit scheduling problem into the existing algorithm but also helps us make it convenient to produce a fitness function.

A5	A1	B6	A4	A7	C2	A8	A4	B5	A3

Table 5.2. fitness table.

5.2.1 Gene Population

There are crossover's happening in genetic algorithm where each gene is broken and connected to another gene at a crossover point to form an offspring. Regulating the number of each categories of employees to a certain number makes it easier for Genetic Algorithm to reach a solution. These tasks that are very problem specific. A basic knowledge of how GA works and how a custom problem is going to react to it helps you to design in a way that it works efficiently.

5.2.2 Mutation

Mutation occurs in the gene where an element of gene is flipped. In GA implementation of our problem, Mutation is designed to identify the type of employee. i.e. L1, L2, L3 and flip the employee within the same category Auditors. Which means that if an element is selected randomly for mutation and it is recognized to be a level 2 auditor, another auditor of level 2 is randomly chosen and replaced. Example:- B2 is flipped to B7. This is done for the same reason so as to maintain the count of each category of employee in the gene. If it was allowed to mutate inter-categorically, there might be a case were the count of each category exceeds the number of required individuals making it harder for the algorithm to converge to a solution.

5.2.3 Fitness Function

The fitness works in the idea of counting conflicts and calculating fitness based on the computed conflicts. The number of Total conflicts is been calculated based on the number of days and audits. First the array is converted to a 2 dimensional array same as in Chapter 3. To do this, a 2 dimensional array is created based on number of days and audits. Then the gene is broken down based on the total count of days and populated into the 2 dimensional array. The 2D array is then scanned to compute conflicts within the array. All the conflicts including the participation of

Level 3 auditors every month, participation of level 2 auditors every week, and load per auditor on each day is been scanned. Total number of conflicts minus computed conflicts is the fitness of the individual gene.

5.2.4 Stopping condition

The algorithm specifies a stopping condition when the solution is reached. Which means that the algorithm stops when an individual of the highest fitness (no conflicts) is found. Apart from this stopping condition, I have added another stopping condition limiting the number of generations to a certain number. The algorithm will stop if it is not able to find a solution in 10000 Generations. (This number can be customized)

CHAPTER 6

DISCUSSION OF RESULTS

6.1 Time

Time efficiency is a crucial part of an algorithm analysis. While analyzing an algorithm's performance the following parameters were kept constant while getting results from both the approaches:

- Days $\rightarrow$ 20
- Audits $\rightarrow$ 8
- L1 auditor Count $\rightarrow$ 5
- L2 auditor Count $\rightarrow$ 7
- L3 auditor Count $\rightarrow$ 5

Method	Time (Milliseconds)
Greedy Approach	3.1
Genetic Algorithm	336

Table 6.1. Time elapsed to get result.

6.2 Space

A memory space of number of audits time number of days is allocated to perform the greedy scheduling approach. Let d be the number of days and let t be the number

of audits that were selected for one execution of the program. The space efficiency is $d \times t$, plus the memory space required to store the audits list and days list.
Contrary to this, genetic algorithm requires a single individual to have a d times t memory space. These genes together form a population. For instance if the population is 50. The memory space is would be $d \times t \times 50$ and over the process of mutation, crossovers and the formation of new generations the space requirements will not get any less or even close to the space required for a solution gained by using greedy approach.

6.3 Elitism Count

Elitism in genetic algorithm (GA) is a method of preserving the elite individuals. Through the process of crossover and mutation the search space is explored forward and in this procedure some elite individuals are altered. To avoid this, a certain number of elite individuals are preserved. The first best chromosome or the few best chromosomes are copied to the new population. The rest is done in a classical way. Such individuals can be lost if they are not selected to reproduce or if crossover or mutation destroys them. This significantly improves the GA's performance [Sivanandam and Deepa (2008)]. For many applications search speed can be greatly improved by not losing the elites. This varies from application to application. The balance between the exploration of the search space and the exploitation of discoveries made within the space is a recurrent theme in GA theory [Coley (1999)].
The accurate count of elitism will be observed in the following chart. Time elapsed to get to solution and the percent chance of getting a solution by varying the elite counts is observed in this section.

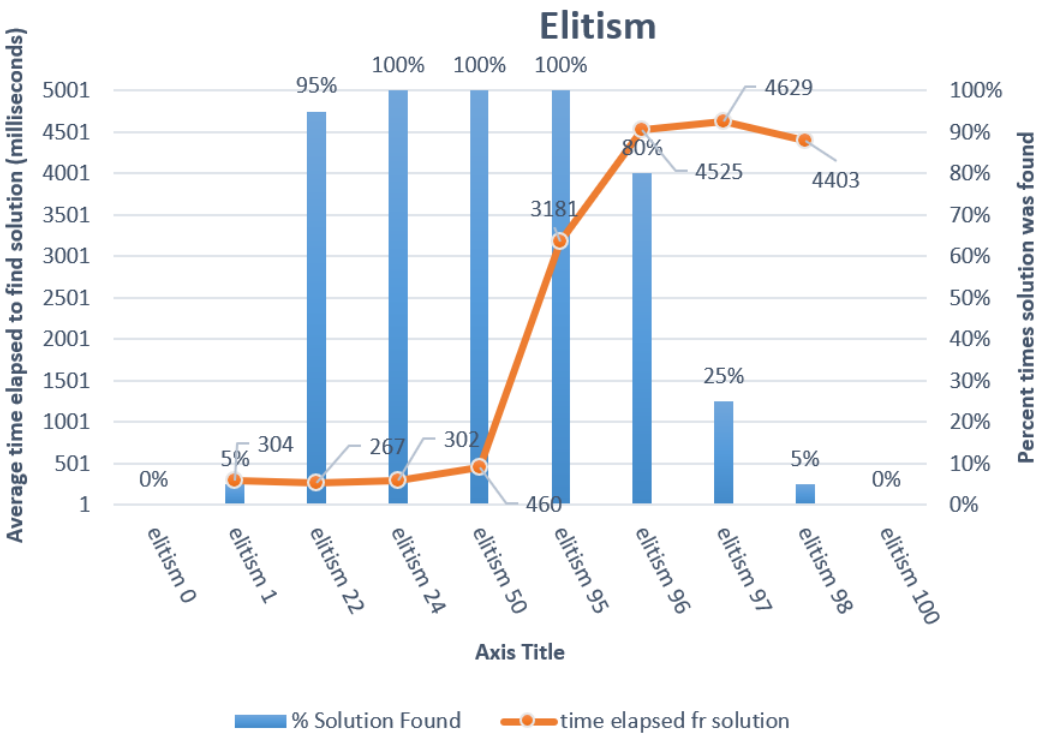

Figure 6.1. Elitism

Figure 6.1 shows dual scale chart of elitism. This chart was obtained by varying elitism count in GA from 0 to count of population (100) and values of percent times solution was found and time elapsed to find a solution was measured. The time elapsed to find a solution, and percent of getting a solution is an average of 100 runs. The line graph in orange represents time elapsed (milliseconds) and the bar chart in blue represents percent times solution was found (%). Please note that all the values of elitism were measured and analyzed but only a few noticeable values are plotted in the above chart in order to make it look clean and readable.

It was compelling to observe that from elitism 24 up until elitism 95 the percent chance of getting of solution remained 100%, However the time elapsed to reach a particular solution changed drastically. A plateau of time elapsed was observed from elitism 1 to elitism 50. A considerable increase of time elapsed was observed as

elitism increased from 50 until elitism 90. GA could not find a solution particularly when elitism was 0, 99 and 100. It was found that for the audit scheduling problem, optimum elitism count considering time and chances of getting solution was 22 elite individuals out of the 100 population.

CHAPTER 7

CONCLUSION

In this paper, various scheduling techniques have been discussed. An audit scheduling problem at BorgWarner has been solved using a greedy approach and a genetic algorithm. The transformation of custom problems and their adaptation to the algorithm and how it is done can be learned from this paper. It was observed that even though genetic algorithm took comparatively more time and space to generate a solution, they did have a higher chance of producing an output. The greedy solution discussed in this paper may get stuck in a trap and might not even shuffle the solution to see for possibilities of reaching the solution. However, this can be improvised by adding random restart and other relevant improvements. The right count of elitism in genetic algorithm for this specific problem found. The best elitism count is 23 of the total 100 population for this problem, which has the increases the chances of converging to a solution within less amount of time. This elitism count however may vary and perform well or bad in other better or worse, depending on the type of problem. Selecting the right elitism count is an important part of genetic algorithm.

BIBLIOGRAPHY

BIBLIOGRAPHY

AIAG 2018 industry (2018), Automotive industry action group.

BorgWarner Inc. (2013), Facts and Figures, BorgWarner Corporate, http://www.borgwarner.com/en/Company/Facts/default.aspx, online; accessed 16 November 2018.

Bulatov, A. A. (2011), Complexity of conservative constraint satisfaction problems, *ACM Trans. Comput. Logic*, 12(4), 24:1–24:66, doi:10.1145/1970398.1970400.

Coley, D. A. (1999), *An introduction to genetic algorithms for scientists and engineers*, World Scientific Publishing Company.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009), *Introduction to algorithms*, MIT press.

CQI2013automotive (2013), *CQI-8 Layered Process Audit Quideline*, Automotive Industry Action Group.

Jacobson, L., and B. Kanber (2015), *Genetic algorithms in Java basics*, Springer.

Koza, J. R. (1992), Genetic programming.

Kroening, D., and O. Strichman (2016), *Decision procedures*, Springer.

Levitin, A. (2012), *Introduction to the design & analysis of algorithms*, Boston: Pearson.

Mitchell, M. (1998), *An introduction to genetic algorithms*, MIT press.

Rao, R. V., P. (Firm), and Y. DDA (2016), *Teaching learning based optimization algorithm: and its engineering applications*, Springer, Cham, New York.

Russell, S. J., and P. Norvig (2016), *Artificial intelligence: a modern approach*, Malaysia; Pearson Education Limited.

Sivanandam, S., and S. Deepa (2008), Genetic algorithm optimization problems, in *Introduction to Genetic Algorithms*, pp. 165–209, Springer.

Van Huyssteen, J. W., and G. V. R. Library (2003), *Encyclopedia of science and religion*, Macmillan Reference USA, New York.